POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

SMBUD Project - Spark

Author(s): **Gabriele Ginestroni**
Giacomo Gumiero
Lorenzo Iovine
Nicola Landini
Francesco Leone

Group Number: **10**

Academic Year: 2022-2023

Contents

1 | Introduction

In this chapter will be presented the problem specification and the hypothesis under which the database is implemented.

2 | Data Structure

2.1. Article structure

In the following `JSON` file we can see how the *Article* is structured in our dataset.
Note: in some fields we decided not to insert all the content only for ease of read reasons.
Note: in `venue`, fields `issue`, `volume` and `publisher` are missing because they are empty in this example.


```
{
"_id": "53e99f86b7602d9702859fdf",
"title": "Locality Sensitive Outlier Detection: A ranking driven
   approach",
"authors": [
    {
    "idAuth": "542a4c9fdabfae61d496694e",
    "org": "Computer Science and Engineering Department, The Ohio State
   University, USA"
    },
    {
    "idAuth": "53f48bc5dabfaea7cd1cce1d",
    "org": "Computer Science and Engineering Department, The Ohio State
   University, USA"
    },
    {
    "idAuth": "53f44b6fdabfaec09f1dd00d",
    "org": "Computer Science and Engineering Department, The Ohio State
   University, USA"
    }
],
"n_citation": 60,
"abstract": "Outlier detection is fundamental to a variety of database
   and analytic tasks. Recently, distance-based outlier detection has
   emerged as a viable and scalable alternative to traditional
   statistical and geometric approaches. In this article we explore the
   role of ranking for the efficient discovery of distance-based
   outliers from large high dimensional data sets. Specifically, we
```

```
    develop a light-weight ranking scheme that is powered by locality
    sensitive hashing, which reorders the database points according to
    their likelihood of being an outlier. We provide theoretical
    arguments to justify the rationale for the approach and subsequently
    conduct an extensive empirical study highlighting the effectiveness
    of our approach over extant solutions. We show that our ranking
    scheme improves the efficiency of the distance-based outlier
    discovery process by up to 5-fold. Furthermore, we find that using
    our approach the top outliers can often be isolated very quickly,
    typically by scanning less than 3% of the data set.",
"doi": "10.1109/ICDE.2011.5767852",
"keywords": [
    "database point",
    "ranking scheme",
    "geometric approach",
    ...
],
"isbn": "978-1-4244-8958-9",
"page_start": "410",
"page_end": "421",
"year": 2011,
"fos": [
    "Locality-sensitive hashing",
    "Anomaly detection",
    "Data mining",
    ...
],
"venue": {
    "raw": "ICDE",
    "type": 0
},
"references": [
    "53e99fddb7602d97028b7e65"
    ]
}
```

2.2. Author structure

In the following `JSON` file we can see how the *Author* is structured in our dataset.
Note: in some fields we decided not to insert all the content only for ease of read reasons.


```
{
"_id":"53f45775dabfaee4dc8162e6",
"name":"Guillermo Jorge-Botana",
"articles":["53e99f86b7602d970285a187"],
"orcid":"0000-0001-5879-6783",
"bio": "Qing-Long Han received the B.Sc. degree in mathematics from the
   ...",
"email":"Guillermo.Jorge-Botana@yahoo.com",
"nationality":"de",
"dob":"1945-06-19T00:00:00Z"
},
```

3 | Data Import

3.1. Setup

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType,
                                    StringType, ArrayType
from pyspark.sql.functions import col, to_timestamp, rand, lit,
                                    collect_list, array

spark = SparkSession.builder \
    .master("local") \
    .appName("SparkByExamples.com") \
    .getOrCreate()
```

3.2. Article Custom Schema

```
schemaArticle = StructType([
    StructField('_id', StringType(), True),
    StructField('title', StringType(), True),
    StructField('authors',
        ArrayType(
        StructType([
            StructField('idAuth', StringType(), True),
            StructField('org', StringType(), True)
        ]), True)
    ),
    StructField('n_citation', IntegerType(), True),
    StructField('abstract', StringType(), True),
    StructField('doi', StringType(), True),
    StructField('keywords', ArrayType(StringType()), True),
    StructField('isbn', StringType(), True),
    StructField('page_start', StringType(), True),
    StructField('page_end', StringType(), True),
    StructField('year', IntegerType(), True),
    StructField('fos', ArrayType(StringType()), True),
```

```
    StructField('references', ArrayType(StringType()), True),
    StructField('venue',
        StructType([
            StructField('raw', StringType(), True),
            StructField('type', IntegerType(), True),
            StructField('issue', StringType(), True),
            StructField('volume', StringType(), True),
            StructField('publisher', StringType(), True)
        ])
    ),
])
```

We decided to use import from schema to explicitly show data structure:

```
df_articles = spark.read.schema(schemaArticle).json("../
                                    dblp_sample_filtered_spark.json",
                                    multiLine=True)
```

The attributes `issue`, `volume` and `publisher` inside venue are moved back in the root structure and removed from the inner struct:

```
df_articles = df_articles.withColumn("issue", col("venue.issue")) \
                         .withColumn("volume", col("venue.volume")) \
                         .withColumn("publisher", col("venue.publisher"))
                                     \
                         .withColumn("venue", col("venue").dropFields("
                                     issue", "volume", "publisher"))
```

3.3. Venue Collection Structure

A new dataframe is created with attributes of venue and the `_id` of the article, then it is all grouped by venue attributes and, for each venue, a list of the articles ids is created. Finally we drop rows with null `raw` to delete inconsistent tuple.

```
df_venues = df_articles.select("venue.raw", "venue.type", "_id") \
                         .groupBy("raw", "type") \
                         .agg(collect_list("_id").alias("artIds")) \
                         .dropna(subset=["raw"])
```

Now we can keep only the raw attribute of the venue:

```
df_articles = df_articles.withColumn("venue_raw", col("venue.raw")).drop
                                        ("venue")
```

In the end, we add a generated field inside venues collection: for each venue a random city is selected that should represent the place where the venue was held.

```
citiesList = ["New York", "London", "Paris", "Berlin", "Madrid", "Rome",
                                         "Dublin", "Copenhagen", "Vienna", "
                                        Amsterdam", "Brussels", "Lisbon", "
                                        Prague", "Athens", "Budapest", "
                                        Warsaw", "Zurich", "Luxembourg", "
                                        Oslo", "Stockholm", "Helsinki", "
                                        Moscow", "Istanbul", "Kiev", "Minsk"
                                        , "Belgrade", "Bucharest", "Sofia",
                                        "Tallinn", "Riga", "Vilnius", "
                                        Tbilisi", "Yerevan", "Baku", "Dubai"
                                        , "Abu Dhabi", "Doha", "Manama", "
                                        Muscat", "Riyadh", "Jeddah", "Mecca"
                                        , "Medina", "Kuala Lumpur", "
                                        Singapore", "Hong Kong", "Shanghai",
                                         "Beijing", "Tokyo", "Seoul", "
                                        Bangkok", "Manila"]
cities = array([lit(city) for city in citiesList])
df_venues = df_venues.withColumn("city", cities.getItem((rand() * len(
                                        citiesList)).cast("int")))
```

3.4. Author Collection Structure

In order to handle authors' collection, we simply import from json with autogenerated schema, applying a conversion from string to timestamp.

```
df_authors = spark.read.json("../dblp_sample_reverted_filtered_spark.
                                        json", multiLine=True)
df_authors = df_authors.withColumn("dateofbirth", to_timestamp(
                                        df_authors["dob"], "yyyy-MM-dd'T'HH:
                                        mm:ss'Z'")) \
                       .drop("dob") \
                       .withColumnRenamed("dateofbirth", "dob")
```